## Acknowledgements

I would like to thank the Department of Justice and the National Institute of Justice for funding support, without which this project would have been impossible to undertake. I would also like to thank Professor David Cyganski for providing me with guidance and insight over the year I spent on this project.

## Abstract

This paper discusses the creation of a demonstration of precision TDOA-based geolocation on an extensible, general-purpose test platform incorporating multiple receivers operating at a wide range of frequencies. Data was collected to show that nanosecond-order TDOA accuracy can be achieved using frequencies in the audio range.

# Contents

**1 Introduction** **10**

1.1 System Architecture . . . . . 11

1.1.1 The Transmitter Signal . . . . . 13

1.1.2 System Overview . . . . . 13

1.2 Project Goal . . . . . 14

1.3 Original Demonstration . . . . . 14

1.4 MQP Goal . . . . . 15

**2 Ultrasonic Transducers** **17**

2.1 Test Configuration . . . . . 17

2.2 Radio Shack Tweeters . . . . . 18

2.3 Ultrasonic Transducer Pair . . . . . 20

2.4 Vifa 40 kHz Tweeters . . . . . 21

2.5 Final Transducer Analysis . . . . . 21

**3 Audio Driver** **25**

3.1 Microphone Preamplifiers . . . . . 25

3.2 Audio Power Amplifier . . . . . 27

3.2.1 Aluminum Reflector . . . . . 27

3.3 Acrylic Resonators . . . . . 28

**4 Data Acquisition Hardware** **31**

4.1 Considerations for Hardware Selection . . . . . 31

4.2 DAQCard Operation . . . . . 32

4.2.1 Analog Input . . . . . 32

4.2.2 Analog Input Timing Signals . . . . . 34

4.2.3 Analog Output . . . . . 36

4.2.4 Analog Output Timing Signals . . . . . 37

4.2.5 General Purpose Counter . . . . . 37

**5 Data Acquisition Software** **38**

5.1 MATLAB Data Acquisition Toolbox . . . . . 38

5.1.1 The Data Acquisition Session . . . . . 39

5.1.2 Advantages and Disadvantages . . . . . 39

5.2 NI-DAQ Driver Set v6.9.3 . . . . . 40

5.2.1 Advantages and Disadvantages . . . . . 41

**6 Audio Demonstration** **42**

6.1 Milestone 1 . . . . . 42

6.1.0.1 Hardware . . . . . 43

6.1.1 Software . . . . . 43

6.2 Milestone 2 . . . . . 46

6.2.1 Hardware . . . . . 47

6.2.2 Software . . . . . 48

6.3 Milestone 3 . . . . . 50

6.3.1 Hardware . . . . . 50

6.3.2 Software . . . . . 51

6.4 Other Revisions . . . . . 52

6.4.1 HTML Statistics Logger . . . . . 52

6.4.2 Graphical User Interface . . . . . 53

6.5 Results . . . . . 54

**7 RF Demonstration** **57**

7.1 Terk Leapfrog LF-30S . . . . . 58

7.2 Data Acquisition . . . . . 60

7.2.1 Throughput of NI-DAQ 6.9.3 Drivers . . . . . 60

7.2.2 NiDAQmx Testing . . . . . 61

7.3 Counter-Trigger Synchronization Method . . . . . 61

7.3.1 Counter-Trigger Test Architecture . . . . . 62

7.3.2 Pulse Trains . . . . . 63

7.3.3 Verification of Counter-Trigger Method . . . . . 63

**8 Conclusions** **65**

8.1 Future Work . . . . . 65

# List of Figures


1.1 Block diagram showing communication between system components. . . . . 12

1.2 Geometric overview of system component placement. . . . . 12

1.3 The OFDM concept. . . . . 13

1.4 Original demonstration, photograph and screen capture with multiple solutions. . . . . 15

2.1 Frequency Response of Radio Shack 25kHz Tweeters (dB) . . . . 19

2.2 Frequency Response of Radio Shack 27kHz Tweeters (dB) . . . . 19

2.3 Frequency Response of Unidentified Radio Shack Tweeters (dB) . 20

2.4 Frequency Response of T/R 25-18Y Ultrasonic Transducers (dB) . 21

2.5 Frequency Response of Vifa 40kHz Tweeters (dB) . . . . . 22

3.1 Block diagram of audio drivers in system. . . . . 26

3.2 Microphone Preamplifier Schematic . . . . . 26

3.3 Power Amplifier Schematic . . . . . 27

3.4 The aluminum cone causes sound to propagate along a horizontal plane. . . . . 28

3.5 Photographs of an acrylic resonator. Left: 3/4 view of resonator. Right: microphone is fit inside a cavity recessed into the tubing and then fit over toroidal base mount. Bottom: overhead view shows interlocking acrylic rings. . . . . 30

4.1 Block diagram of DAQCard 6062-E operation. (Reprinted with permission of National Instruments Corporation. All rights reserved.) 33

4.2 Analog input block diagram. (Reprinted with permission of National Instruments Corporation. All rights reserved.) . . . . . 34

4.3 TRIG1 timing diagram. (Reprinted with permission of National Instruments Corporation. All rights reserved.) . . . . . 35

4.4 CONVERT* timing diagram. (Reprinted with permission of National Instruments Corporation. All rights reserved.) . . . . . 35

4.5 Timing sequence of TRIG1, STARTSCAN, and CONVERT*. (Reprinted with permission of National Instruments Corporation. All rights reserved.) . . . . . 36

6.1 Milestone 1: two Rx tracking one Tx. . . . . 44

6.2 Connections to the SCB-68 connector block. Note that the configuration is flexible for a 2 Tx, 1 Rx or 1 Tx, 2 Rx demonstration. . 47

6.3 Photograph of the Milestone 3 configuration. . . . . 50

6.4 Milestone 3 signal connections. Note the addition of two analog input channels and the removal of an analog output. . . . . 51

6.5 Output of the HTML statistics generator. . . . . 53

6.6 Screen capture of the audio locator GUI. . . . . 54

7.1 Frequency response of the LeapFrog A/V transceiver system. . . . 59

7.2 Block diagram of counter-trigger control configuration. . . . . 64

# List of Tables

2.1 Audio/RF Correspondences . . . . . 24

6.1 Statistical data from 15 runs of the Milestone 3 demonstration at 1000 cycles per run. . . . . 55

# Chapter 1

# Introduction

On December 3, 1999, six firefighters died in a warehouse fire in Worcester, MA. Due to the lack of any visibility whatsoever inside the warehouse, two firefighters and the search teams that entered the building to rescue them perished, literally 100 feet from safety. This posed a question to engineers: how can technology be developed to prevent this kind of tragedy from happening in the future?

The Worcester cold storage fire underscored the importance of precision positioning information to the men and women who risk their lives by being the first people on the scene of a disaster. Not only do first responders require position information, but any technology that they use needs to be quickly deployable so that time is not spent on equipment calibration but rather spent on the urgent task at hand.

A precision positioning system that quickly and automatically configures to new environments has the potential to help not only firefighters, but law enforcement officers, military responders, and corrections officers. The Department of Justice and the National Institute of Justice provided our group with funding for the creation of a demonstration based on techniques documented in papers by Cyganski,

et al.[1]

## 1.1 System Architecture

At its core, the Precision Personnel Locator consists of a number of reference nodes, placed outside a target area, receiving and collaborating with one another the data from transmitters on the personnel inside a target area. The reference nodes send received data to a command and control unit, which uses that data to extrapolate the position of each transmitter with a high degree of accuracy. See **Figures 1.1** and **1.2** for depictions of the reference node and transmitter configuration.

The reference nodes are all synchronized with each other, and they collaborate time-of-arrival (TOA) data relative to a master clock to obtain time-difference-of-arrival (TDOA) data. Using receiver-relative TDOA data means that the transmitters need not be synchronous with the rest of the system, which leads to greatly simplified circuitry on the transmitter side. Thus the complex equipment can remain outside of the target area of the emergency.

The transmitter position is estimated using the TDOA data and the positions of the reference nodes relative to an arbitrarily-defined origin. Thus reference nodes can be placed anywhere outside a target area. The command and control unit can be defined as an origin, and the relative positions of the reference nodes can be determined using a positioning algorithm specifically used to determine node positions. This process defines an ad-hoc coordinate system to be used in all further calculations.

[1]Cyganski, D., et al. *A Multi-Carrier Technique for Precision Geolocation for Indoor/Multipath Environments.* ION GPS/DNSS 2003 Conference, September 9-12, 2003, Portland, USA.
Cyganski, D., et al. *Performance of a Precision Indoor Positioning System Using a Multi-Carrier Approach.* ION National Technical Meeting 2004, San Diego, California, January 26-28 2004.

Figure 1.1: Block diagram showing communication between system components.

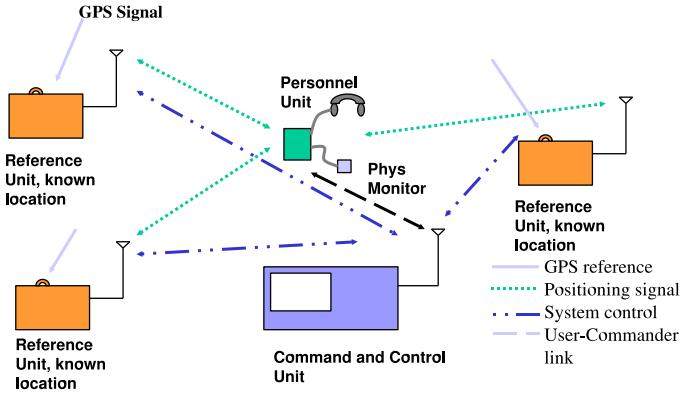



Figure 1.2: Geometric overview of system component placement.

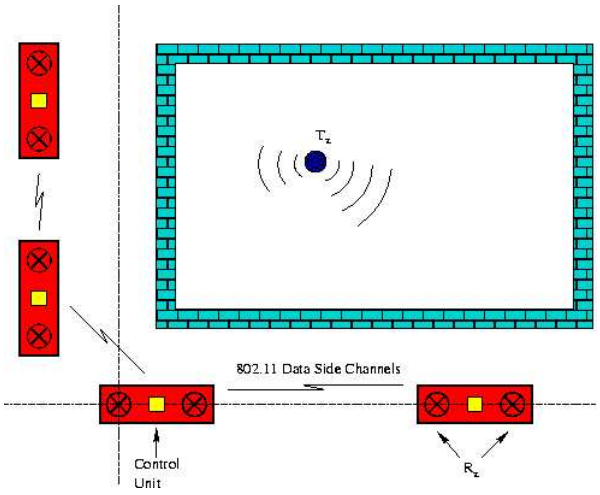
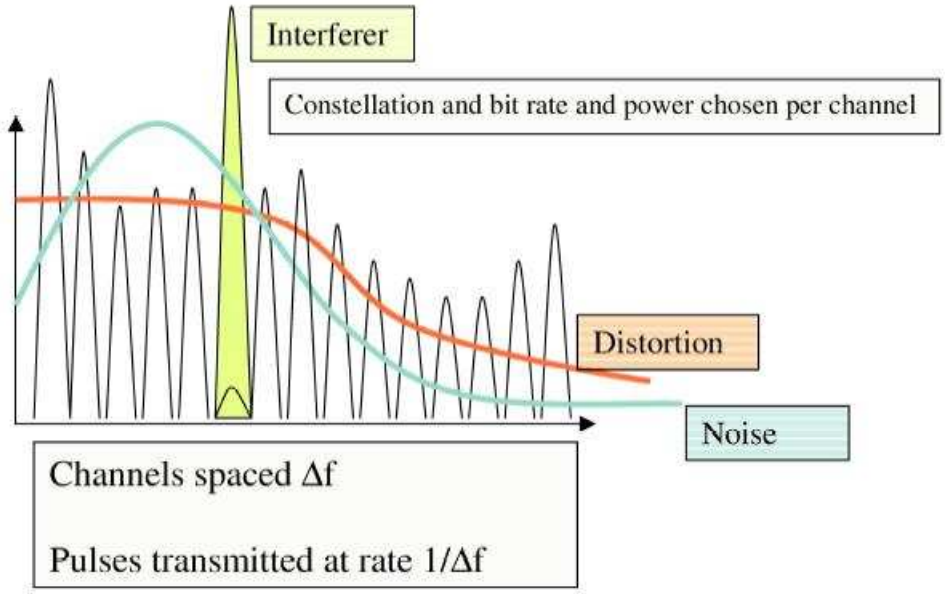


Figure 1.3: The OFDM concept.




### 1.1.1 The Transmitter Signal

The transmitters generate a periodic orthogonal frequency division multiplexed (OFDM) signal. OFDM is a digial modulation technique which typically involves sending modulated carriers in parallel with frequency spacing $\Delta f$ such that $\Delta f$ is the minimum frequency spacing required for the signal to satisfy orthogonality (and hence, detectability) conditions. See **Figure 1.3** for an illustration of an OFDM signal.

### 1.1.2 System Overview

The Precision Personnel Locator will

- estimate the location of a user in three dimensions relative to a chosen reference point with approximately 2 cm of accuracy.
- provide a graphical display of user positions and paths taken at a command station.
- provide an audio communication link with users so that a commander can issue self-rescue instructions and other vital information.
- operate in at least the range of a city block.

Possible extensions to the system include the addition of physiologic telemetry so commanders can keep track of vital signs of users, as well as the addition of Geographic Information Systems (GIS) data overlays.

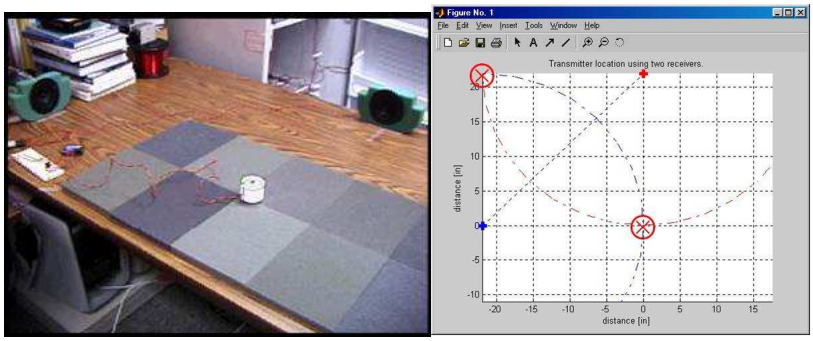
## 1.2 Project Goal

The goal of the Precision Personnel Locator project is to create technology to aid and rescue those who put the mission of our safety before their personal safety.

## 1.3 Original Demonstration

An initial demonstration was constructed in the winter of 2001 by Prof. David Cyganski and Joshua Resnick at Worcester Polytechnic Institute. The demonstration consisted of two stationary speakers transmitting the OFDM signal to a single receiver. The OFDM signal consisted of 100 carriers spaced between 2 kHz and 7.5 kHz. Both of the speakers and the microphone were connected to the sound card of a tower PC, and both were controlled using MATLAB's Data Acquisition Toolbox.

Figure 1.4: Original demonstration, photograph and screen capture with multiple solutions.




The position of the microphone was correctly tracked within a specific test area, and the algorithm correctly identified located obstacles creating reflections near the microphone. Also, due to the number of reference nodes, the algorithm solved for position using TOA rather than TDOA data.

Although the positioning solutions were correct, they were certainly not accurate, as the predicted location of the microphone experienced roughly 2 cm of "jitter." Nor was the solution unique. Due to the presence of only two reference nodes, the algorithm yielded either 0, 1, or 2 solutions, depending on the location being solved for (see **Figure 1.4**).

## 1.4 MQP Goal

The original demonstration consisted of a two-dimensional, TOA-based solution limited to 2 transmitters and 1 receiver at the audio transmission range. These limitations were based on the platform on which the demonstration was implemented: a consumer PC sound card. The purpose of this MQP was to migrate

the demonstration to a more extensible, general-purpose test platform so that the demonstration could incorporate multiple receivers operating at a wide range of frequencies, could be executed across multiple computers, could be synchronized to an externally generated clock signal, and could be extended to three dimensional positioning.

# Chapter 2

# Ultrasonic Transducers

As we already had a set of speakers and microphones that worked in the 2 kHz to 8 kHz range, we needed to find a set of ultrasonic transducers that could be used to create an ultrasonic version of the demonstration. A wide range of transducers were examined, from commercial tweeters for home stereo systems to expensive ultrasonic-specific tweeters. The tests were conducted looking for a set of transducers with a -3 dB bandwidth of at least 6 kHz where the lower bound of the bandwidth was greater than 20 kHz. However, a -10 dB bandwidth of 6 kHz would also be acceptable, as within this range the signal can be amplified in software with an acceptable increase in noise levels.

## 2.1 Test Configuration

All of the transducers were tested using one of two methods.

In the case of symmetric sets (i.e., no manufacturer differentiation between a transmitter and a receiver), two transducers of the same kind were designated as trans-

mitter and receiver. The transmitter input was connected to a function generator and to channel 1 of an oscilloscope, and the receiver ouput was hooked up to channel 2 of the oscilloscope. The transmit/receive pair were acoustically coupled. The function generator was manually set to a sine wave of a certain frequency as measured on channel 1, and set to 1 V or 2 V peak-to-peak amplitude, depending on the transducers being measured. The peak-to-peak voltage of the received signal on channel 2 was recorded, and the function generator was advanced manually to the next desired frequency.

In the case of asymmetric sets (i.e., where the manufacturer designated a transmitter and a receiver), the same method was used except care was taken to use the given devices as they had been designated by the manufacturer.

Recorded data was entered into MATLAB, and the following frequency response plots were generated.

## 2.2 Radio Shack Tweeters

There were three Radio Shack brand tweeters tested for their ultrasonic response. Although not much performance was expected from the tweeters in terms of bandwidth, they were cheap and easy to acquire, and as such at least provided a comparison point to see if more expensive specialized equipment was truly worth the cost.

**Figure 2.1** shows the response of a 25 kHz-rated tweeter to a 2 V sinusoid of varying frequency. The peak response occurs at 19.5 kHz, with a -3 dB bandwidth of about 500 Hz.

The 27 kHz-rated tweeter in **Figure 2.2** shows a more pronounced peak at 24 kHz,

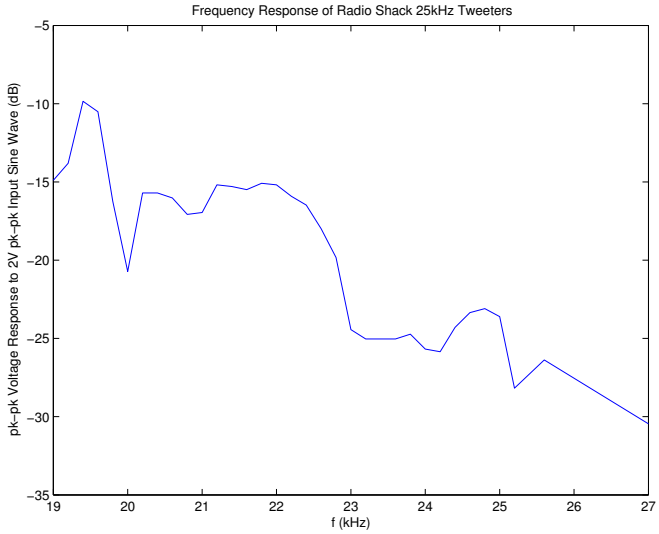



Figure 2.1: Frequency Response of Radio Shack 25kHz Tweeters (dB)

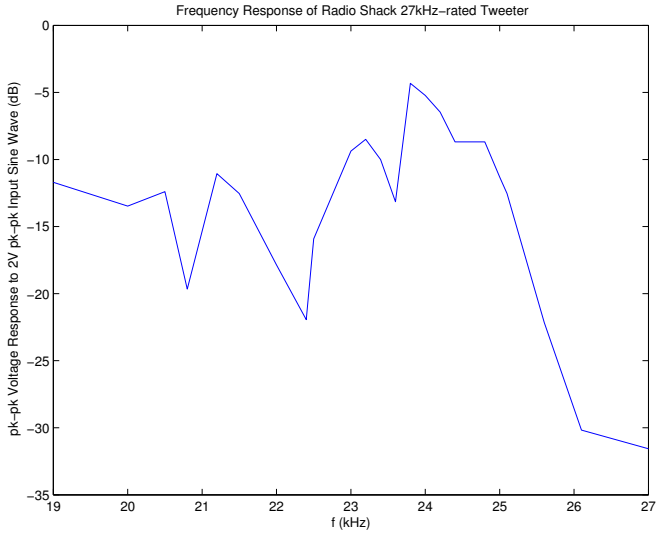



Figure 2.2: Frequency Response of Radio Shack 27kHz Tweeters (dB)

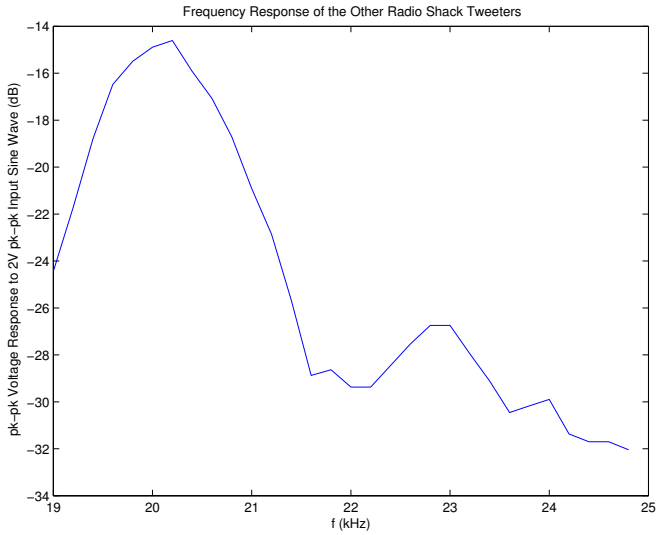



Figure 2.3: Frequency Response of Unidentified Radio Shack Tweeters (dB)

again subjected to a 2 V sinusoid. Although the -3 dB bandwidth is again about 500 kHz, the -10 dB bandwidth ranges from 23 kHz to 25.5 kHz.

Finally, **Figure 2.3** shows the response to a 2 V sinusoid of a pair of Radio Shack tweeters of unknown specification (the tweeters lacked any accompanying documentation). The response of these tweeters peaked around 20.5 kHz with a -3 dB bandwidth of 1 kHz. The -10 dB bandwidth ranges from 19 kHz to 21.5 kHz.

## 2.3 Ultrasonic Transducer Pair

A more expensive model of transducer tested was a pair of T/R 25-18Y ultrasonic transducers. This pair of tranducers was asymmetric, with T 25-18Y designating the transmitter and R 25-18Y designating the receiver. The tests yielded about 700 Hz of bandwidth between the -3dB points, with peak response at 25.6 kHz.

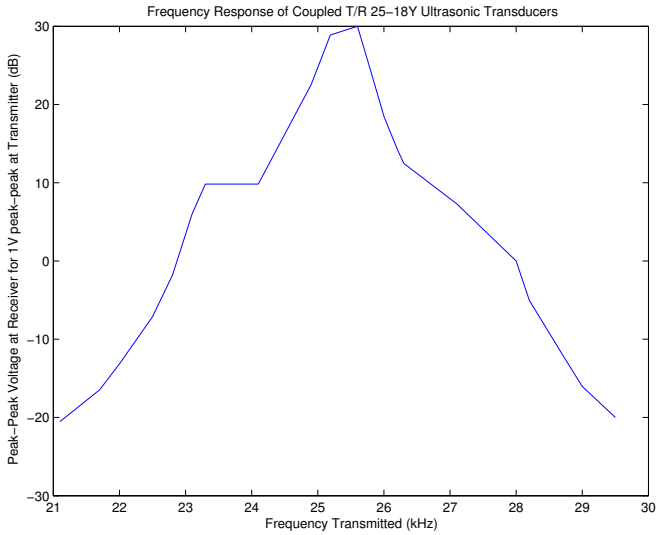



Figure 2.4: Frequency Response of T/R 25-18Y Ultrasonic Transducers (dB)

Although this bandwidth performance was unacceptable, at least the frequency response (seen in **Figure 2.4**) showed that the parts performed to their specification.

## 2.4 Vifa 40 kHz Tweeters

The most expensive pair of transducers tested was a pair of symmetric Vifa 40 kHz tweeters. As seen in **Figure 2.5**, measuring from our defined lower bound of 20 kHz, the -3dB bandwidth was 40 kHz, which was exactly in the desired frequency range.

## 2.5 Final Transducer Analysis

If we were to use any ultrasonic transducers, the Vifa set, with its excellent frequency characteristics, would certainly have been the one to use, as its perfor-

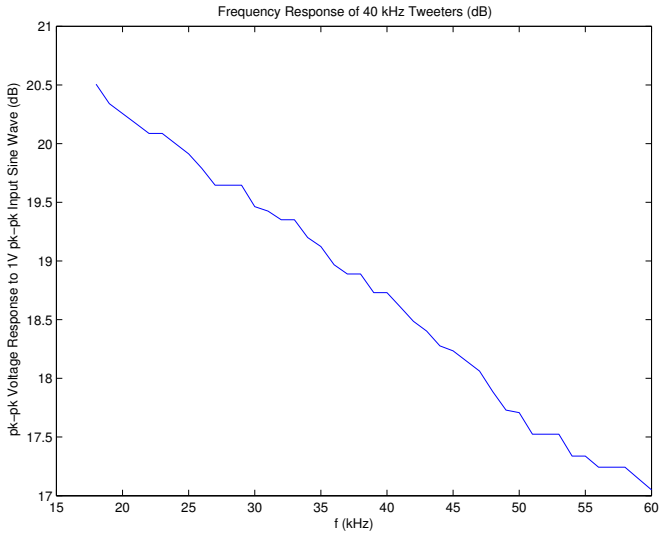



Figure 2.5: Frequency Response of Vifa 40kHz Tweeters (dB)

mance justified its price tag at $200 a pair. Ultimately, we chose not to construct an ultrasonic-based test system. The choice was made on the basis of the analog that could be established between an audio frequency wavelength in air and an RF wavelength through vaccuum.

The accuracy of the location algorithm is based, in part, on the wavelength of the transmitted location signal, and wavelengths depend on both the frequency of a signal and the medium through which a signal is sent. While audio signals require a medium of propagation, such as air or water, RF signals propagate through vaccuum. Although RF signals oscillate at much higher frequencies than audio signals, the physical wavelength of an audio signal through air corresponds to the physical wavelength of a higher frequency RF signal through vacuum.

This relationship is best illustrated by the equation

$$speed = frequency \cdot wavelength$$

Sound travels at 343 m/s in air, while an RF signal travels through vaccuum at the speed of light, $2.998 \cdot 10^8$m/s. We can substitute the speed of sound along with an arbitrary audio frequency to get a wavelength. We can substitute the wavelength back into the equation with the speed of light to get the corresponding radio frequency. For example, the lowest frequency used in our audio demonstration was 2.16 kHz. The following equations yield the corresponding radio frequency for that wavelength through vaccuum:

$$wavelength = \frac{speed}{frequency} = \frac{343(m \cdot s^{-1})}{2160(s^{-1})} = .1588(m)$$

$$frequency = \frac{speed}{wavelength} = \frac{2.998 \cdot 10^8(m \cdot s^{-1})}{.1588(m)} = 1.88{\cdot}10^9(s^{-1}) = 1.88(GHz)$$

Using these equations, we can calculate the data shown in **Table 2.1**, which contains the lower and upper frequency bounds of the signals used in the audio demonstration versus the upper and lower bounds of a comparable RF system. The 5 kHz-wide signal used in the audio demonstrations corresponds to a 5 GHz ultra-wideband signal. Hence by using audio frequencies in the 2.16 kHz to 7.54 kHz range, we attain a 1:1 scale model of an RF system in our target band. The only necessary change in the location algorithm is the replacement of the speed of light by the speed of sound in air when solving for position from the TDOAs.

Table 2.1: Audio/RF Correspondences

| Audio Freq. in Air | Wavelength | Radio Freq. in Vaccuum |
|---|---|---|
| 2.16 kHz | .1588 m | 1.88 GHz |
| 7.54 kHz | .0455 m | 6.57 GHz |

# Chapter 3

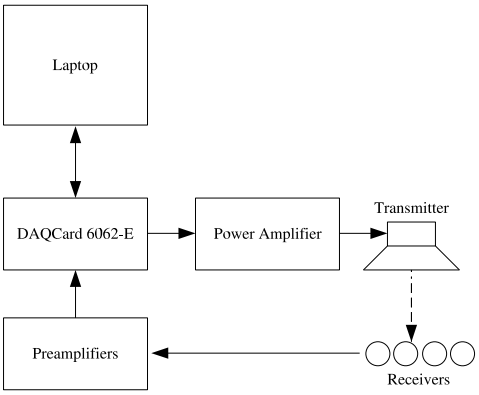
# Audio Driver

As the data aquisition card contains no onboard driver circuitry, it was necessary to design and to implement preamplifiers for the microphones and a power amplifier for the speaker.

## 3.1 Microphone Preamplifiers

The variable-gain preamplifiers, as shown in **Figure 3.2**, is a simple non-inverting op-amp configuration using the LM324, a commonly available single-supply op-amp. Of note is the voltage divider, which creates a reference voltage of $\frac{V_{cc}}{2}$, or 2.5 V in this case. By referencing the op-amp to this voltage, the output of the preamplifier maintains a DC offset of $\frac{V_{cc}}{2}$, thus "centering" the output and preventing clipping from occurring.

Figure 3.1: Block diagram of audio drivers in system.



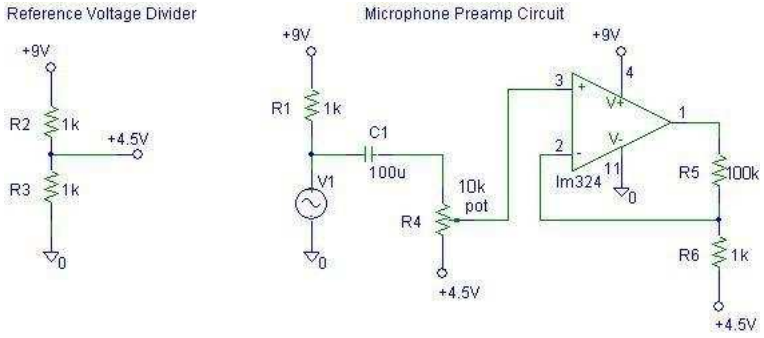




Figure 3.2: Microphone Preamplifier Schematic

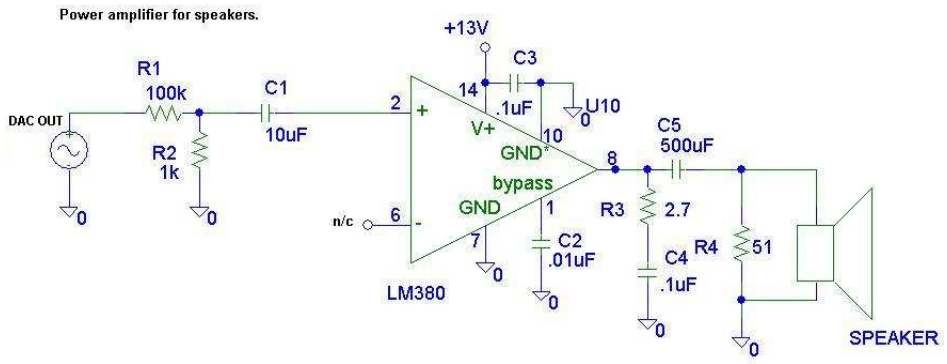



Figure 3.3: Power Amplifier Schematic

## 3.2 Audio Power Amplifier

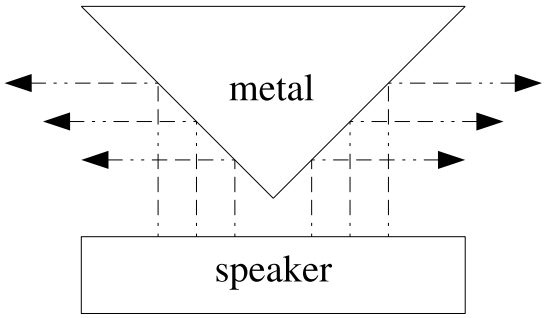
The audio power amplifier is based on the LM380, a fixed-gain, 2W audio power amp. All details about specific connections can be gathered from the schematic in **Figure 3.3**, but there are some physical constraints that are of note.

We discovered, after testing initial circuit designs, that the output of the LM380 tended to oscillate rapidly between the supply rails. After some troubleshooting, it was determined that this instability was due to feedback caused by power supply coupling. To correct for this problem, we had to keep all connecting wires as short as possible. In addition, we required 100uF capacitors on the power bus at pin 14 (V+); by placing the capacitors on the power supply near the supply pins for the IC, we ensured an AC ground at the chip.

### 3.2.1 Aluminum Reflector

In order to ensure that sound generated by the speaker (which faces up away from the table) propagates towards the microphones in the manner of a two-dimensional

Figure 3.4: The aluminum cone causes sound to propagate along a horizontal plane.




omnidirectional point source, a 45-degree metal cone was crafted from aluminum sheeting. The cone is suspended over, but not touching, the center of the speaker. See **Figure 3.4** for an illustration.

## 3.3 Acrylic Resonators

The microphones were found to be extremely directionally sensitive, so a set of audio resonators was built to alleviate this problem.

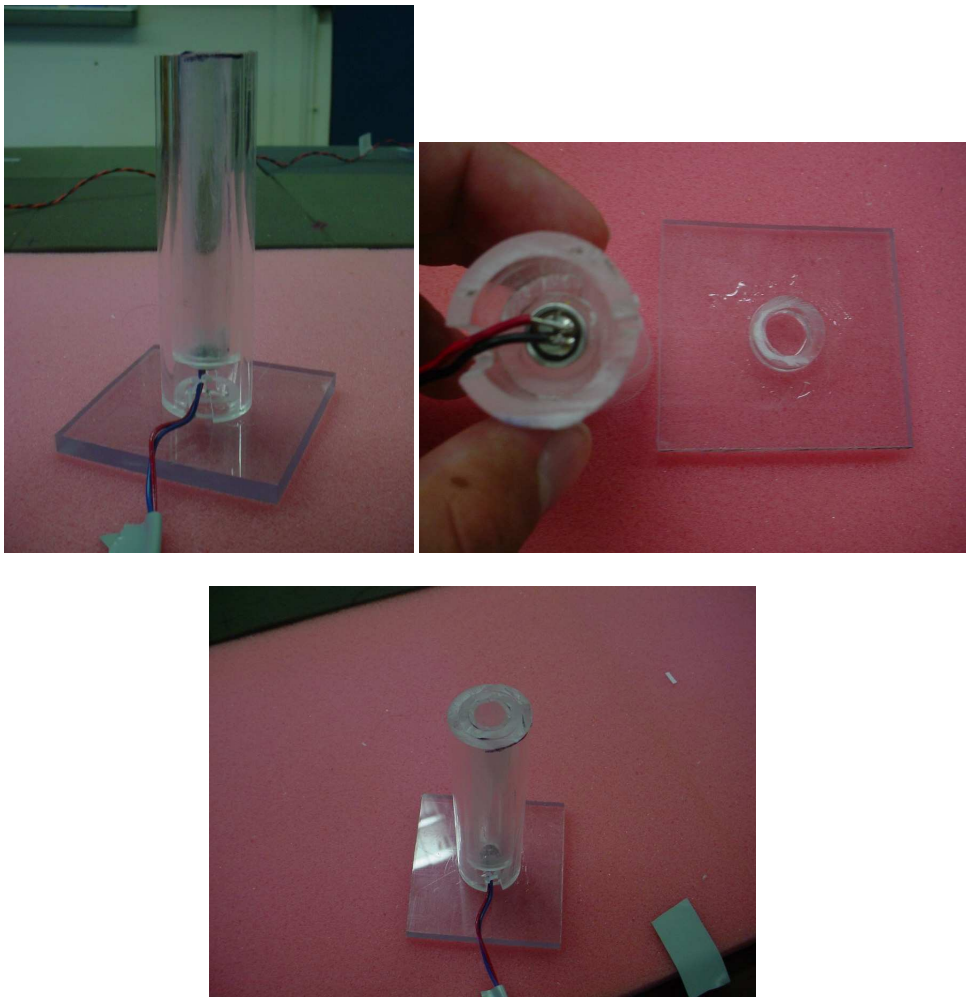
The acrylic resonators were made from acrylic tubing acquired from the ECE Shop. They consist of three tubes of decreasing diameter, one fitting tightly inside the other. The smallest tube has an internal diameter of just under 3/8". The microphone's diameter is just over 3/8", so after a little bit of filing done on the internal aperture, the microphone fit snugly inside. The internal two tubes are cut to 3", whereas the outer tube is 3 1/2". This gives 1/2" of space inside for the base mount and the wires. A notch was cut into the side of the resonator, which provides room for external wires to reach the internally-mounted microphone.

Sound that passes over the top side of the acrylic tube resonates at the frequencies that need to be amplified (the frequency is determined by the length of the shaft created by the hole), which creates a physical band-boost filter, in effect amplifying sound via resonance at around 5 kHz.

More importantly than applying a physical band-boost filter, the resonators caused the received signal to be omnidirectionally phase coherent. The microphone registered the sound passing over the resonator opening from any direction.

For stability, each resonator was equipped with a base. The base is a rough-cut 2" square of 1/4" thick Lexon with a mounting toroid affixed to it with epoxy. The toroid is a 1/8" cut of the second-largest acrylic tube. This allows the largest tube to fit tightly over it, creating a base that can be fixed to the table. If a microphone needs servicing, the microphone unit can be removed from the base without compromising the position in which it had previously been set. For photographs of the final result, see **Figure 3.5**.

Figure 3.5: Photographs of an acrylic resonator. Left: 3/4 view of resonator. Right: microphone is fit inside a cavity recessed into the tubing and then fit over toroidal base mount. Bottom: overhead view shows interlocking acrylic rings.



# Chapter 4

# Data Acquisition Hardware

Although the original demonstration used a standard PC sound card to handle signal I/O, sound cards are not designed to handle data acquitision and waveform generation at rates much higher than 25 kHz. The best consumer sound cards available can sample signals at 46 kHz; but as we planned for a robust system that could be tested in a wide range of frequencies, we had to look at hardware designed specifically for PC-based data acquisition.

## 4.1 Considerations for Hardware Selection

The major deciding factor in choosing our data acquisition hardware was the fact that our demonstration system needed to be portable; that is, we required mobility to demonstrate the rapid-deployment capabilities of the final system. In order to accomplish this, we envisioned an ideal demonstration involving 5 receivers and 1 transmitter, with each reference node connected to its own laptop. Since the sensors would be interfaced to the data acquisition hardware and then to the laptop,

a PCMCIA bus solution seemed the most appropriate choice for interfacing with a laptop.

After conducting research, we concluded that the only company currently offering data acquisition hardware for a PCMCIA bus at the analog output and input transfer rates we required was National Instruments, and the specific piece of hardware we purchased was the DAQCard 6062-E.

## 4.2 DAQCard Operation

The National Instruments DAQCard 6062-E is "a multifunction analog, digital, and timing I/O" data acquisition device. The DAQCard "features a 12-bit A/D converter (ADC), two 12-bit D/A converters (DACs), eight lines of TTL-compatible digital I/O (DIO), and two 24-bit counter/timers for timing I/O (TIO)."

For a block diagram of the DAQCard, see **Figure 4.1**.

### 4.2.1 Analog Input

The DAQCard features 16 analog input lines at 12-bit resolution, which can be configured as 16 referenced single-ended (RSE) or 8 differential inputs. Since our microphone preamplifiers are single-supply, we use the analog inputs in RSE mode. The input voltage range is programmable, but can be chosen from several sets of ranges between -10 V and +10 V in single-ended mode.

The ADC which samples the analog input lines can operate at conversion rates of up to 500/N ksamples/sec, where N is the number of channels being "simultaneously" sampled. This is because the DAQCard contains only one ADC onboard

Figure 4.1: Block diagram of DAQCard 6062-E operation. (Reprinted with permission of National Instruments Corporation. All rights reserved.)

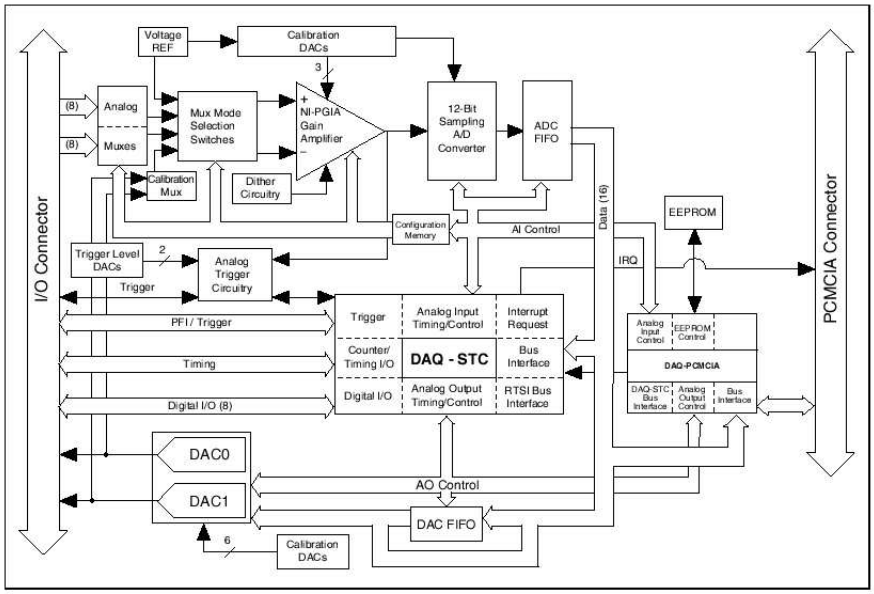
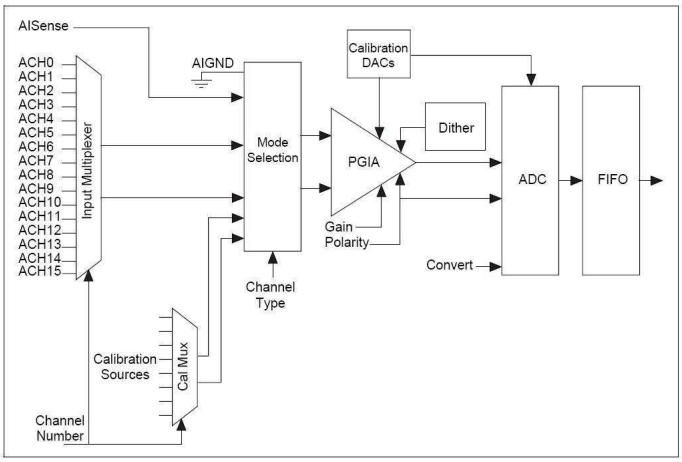


Figure 4.2: Analog input block diagram. (Reprinted with permission of National Instruments Corporation. All rights reserved.)




for all the channels, so true simultaneous sampling is not possible; the channels are multiplexed into the ADC.

See **Figure 4.2** for a block diagram of the analog input subsystem.

### 4.2.2 Analog Input Timing Signals

Of the 8 timing signals available on the DAQCard 6062-E for control and analysis of analog input, only 3 were used in this project. Those signals include TRIG1, CONVERT*, and STARTSCAN.

The TRIG1 signal, as used in our project, initiates the data acquisition sequence. This can mean different things in different situations. If the DAQCard is configured for a single-buffer acquitision of 1000 samples, then TRIG1 initiates the one-shot acquisition. In the case of a double-buffered acquisition, then TRIG1 initiates the continuous waveform acquisition, which operates indefinitely until a stop com-

Figure 4.3: TRIG1 timing diagram. (Reprinted with permission of National Instruments Corporation. All rights reserved.)

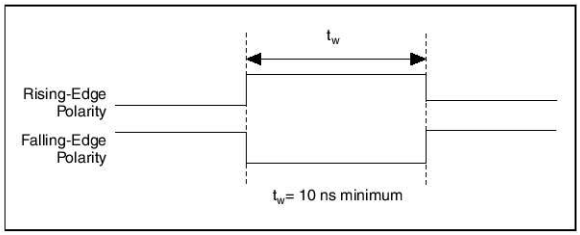
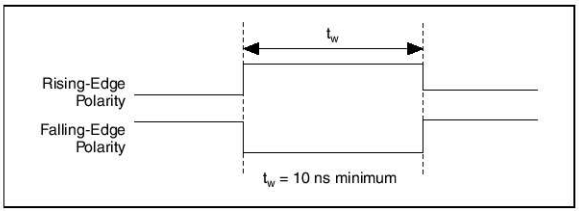



Figure 4.4: CONVERT* timing diagram. (Reprinted with permission of National Instruments Corporation. All rights reserved.)

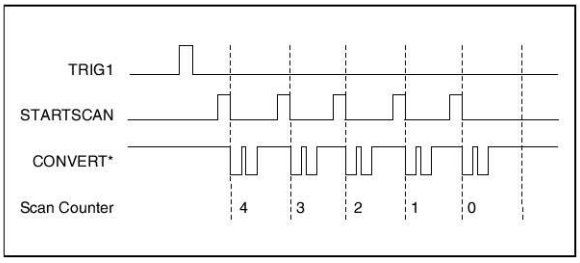



mand is issued. TRIG1 is edge-triggered, and may be configured as rising-edge or falling-edge in software. See **Figure 4.3** for timing data.

The CONVERT* signal controls the rate at which data aqcuisition occurs. After TRIG1 is initiated, the rising or falling edge (depending on user configuration) of CONVERT* causes the ADC to convert the analog input to a digital value and send it to the onboard FIFO to be later read by the PC (see **Figure 4.2**). See **Figure 4.4** for timing data.

The STARTSCAN signal is relevant when there is more than one analog input to be read and when CONVERT* is internally generated. For N channels to be read, each rising or falling edge of STARTSCAN initiates N pulses of the CONVERT*

Figure 4.5: Timing sequence of TRIG1, STARTSCAN, and CONVERT*. (Reprinted with permission of National Instruments Corporation. All rights reserved.)




signal, at whatever speed CONVERT* is internally configured. Only on the next active edge of STARTSCAN will CONVERT* pulses be generated again. For a depiction of this sequence of events, see **Figure 4.5**.

The most important feature of these signals is that they may be

- generated internally and viewed externally on an oscilloscope, or
- generated externally and routed to the internal control lines.

The former feature allows for analysis and troubleshooting, as well as exportation of timing signals to control other devies; the latter feature means that timing signals can come from external control sources, allowing for multiple-device synchronization. For an example of such synchronization, see **Section 7.3**.

### 4.2.3 Analog Output

The DAQCard 6062-E provides two channels of analog output at 12-bit resolution. Waveforms may be updated at rates as high as 500 ksamples/sec, but like the analog inputs, this rate is cut in half when both channels are being used simultaneously.

### 4.2.4 Analog Output Timing Signals

Of the three timing signals available for analog output, only one was used during the project. The UPDATE* signal is very much like the CONVERT* signal, except it controls the rate at which the DAC is updating its output. Since the transmitter in our system is asynchronous, the UPDATE* signal was never externally controlled, and was only used as a timing reference for the input conversion rate of other DAQCard units, as explained in **Section 7.3**.

### 4.2.5 General Purpose Counter

The DAQCard comes with the DAQ-STC (System Timing Control) application-specific integrated circuit (ASIC), with seven 24-bit counters and three 16-bit counters. While almost all of these counters are reserved for use by the analog input and analog output subsystems, the DAQCard leaves two 24-bit registers for general use. These general purpose counters are referred to as GPCTR0 and GPCTR1, and can be accessed through a family of **gpctr** functions via the NI-DAQ driver set (see **Section 5.2** for a description of the software drivers).

For a detailed description of general purpose counter usage in this project, see **Section 7.3**.

# Chapter 5

# Data Acquisition Software

The DAQCard 6062-E can be controlled several ways. National Instruments provides a set of dynamically linked libraries (DLLs) with C++ functions that can be called from any program that includes the libraries. In addition, MATLAB has a Data Acquisition Toolbox which supports the NI-DAQ 6.9.3 driver set and NI E-series devices. Initially, we chose to use the Data Acquisition Toolbox.

## 5.1 MATLAB Data Acquisition Toolbox

The Data Acquisition Toolbox is a set of M-files and DLLs which provide the user with "a convenient front end to the driver software" for a given hardware device[1]. While the user interacts with the toolbox, the toolbox is passing data to and receiving data from the device drivers. The device drivers communicate directly with the hardware, in the background.

[1]Data Acquisition Toolbox documentation. © 1994-2004 The MathWorks, Inc.

### 5.1.1 The Data Acquisition Session

Any program which requires data from an external device, be it a PC sound card or a multifunction DAQ, has a program flow as follows:

1. Initialization: device objects are created.
2. Configuration: object properties are written such that number of I/O channels, voltage ranges, sampling frequencies, etc., are set to desired values.
3. Execution: device objects are armed, at which point functionality may be triggered internally or externally.
4. Termination: device objects are deleted.[2]

A device object behaves like any object in MATLAB. An object constructor can be called to create a device object, which then has certain properties depending on which constructor was used to create the object. Three kinds of objects are available for use in the Data Acquisition Toolbox: **analoginput**, **analogoutput**, and **digitalio**. Each object type is associated only with one particular subsystem on the hardware device with which it interfaces. Of the three object types listed, only the first two were used in this project.

### 5.1.2 Advantages and Disadvantages

The major advantage to using the Data Acquisition Toolbox is that it is already integrated with MATLAB. Also, the first iteration of the demonstration (as described in **Section 6.1**) used the Data Acquisition Toolbox for audio I/O using the

[2] *Data Acquisition Toolbox Quick Reference Guide*. © 1994-2004 The MathWorks, Inc.

PC sound card, therefore the toolbox made the code transition between Milestone 1 and Milestone 2 almost seamless.

The key disadvantage to the use of the Data Acquisition Toolbox is its lack of flexibility. The toolbox does not allow for external timing signal control, nor does it allow for software-controlled routing of internal signals, which became necessary features during our RF functionality tests, as explained in **Section 7**.

## 5.2 NI-DAQ Driver Set v6.9.3

NI-DAQ v6.9.3 "is a set of functions that control all of the National Instruments plug-in DAQ devices for analog I/O, digital I/O, timing I/O, SCXI signal conditioning, and RTSI multiboard synchronization."[3] Every command issued to the DAQCard must be passed to the board through these drivers. Although the Data Acquisition Toolbox as available in MATLAB does provide its own front-end, any commands issued through the toolbox are passed to NI-DAQ functions which control the device directly. National Instruments provides the NI-DAQ functions as a set of C++ functions that can be called directly from a DLL.

MATLAB 6.5 R13 allows the access of functions defined in Windows standard DLLs through the generic DLL package, which provides access to three new MATLAB functions: **loadlibrary**, **unloadlibrary**, and **calllib**. The first two functions load an unload the functions in a DLL, while **calllib** accepts as arguments the library name, a function name, and any variables to be passed to a function. For example, calling a function "add," which returns the summation of its two variables, from FOO.DLL would involve the following code in an M-file:

```
loadlibrary('foo.dll','foo.h');
```

[3] *NI-DAQ User Manual for PC Compatibles.* National Instruments, October 2000.

```
result = calllib('foo','add',2,3);
unloadlibrary('foo');
```

Using MATLAB's generic DLL package allowed us to call low-level NI-DAQ functions directly.

### 5.2.1 Advantages and Disadvantages

The advantage of calling the driver functions directly is the flexibility afforded by register-level control of the DAQCard. The NI-DAQ drivers allow for

- software routing of any and all DAQCard signals.
- external synchronization and control of CONVERT*, UPDATE*, and other signals described in **Section 4.2**.
- greater control of software flow due to higher granularity of process control.

The greatest disadvantage of using the NI-DAQ drivers is their complexity relative to the MATLAB Data Acquisition Toolbox. The NI-DAQ implementation of a single **getdata** command in the toolbox requires roughly five function calls and the creation of a **while** loop.

# Chapter 6

# Audio Demonstration

As explained in **Section 1.3**, the original audio demonstration used two transmitters and one receiver connected to a sound card. The location of the receiver was tracked around the test area, which measured 22" x 22".

The audio demonstration was built over the course of a year, from Summer 2003 to Spring 2004, as a series of milestone designs. Each milestone represents a significant change in the hardware configuration and functionality, and as such, this chapter covers the milestones in chronological order beginning where the original demonstration (see **Section 1.3**) left off.

## 6.1 Milestone 1

The first new iteration of the demonstration was completed in July of 2003. The purpose of this iteration was to confirm the theory that the location algorithm would still work if the function of the receivers and transmitters were reversed (i.e., fixed receivers tracking the position of a moving transmitter).

#### 6.1.0.1 Hardware

This iteration of the indoor geolocator involved two receivers (audio microphones) and one transmitter (audio speaker). The two receivers remained at fixed locations, each on an opposite diagonal corner of a 22 inch square. The two receivers were connected to the left and right channels of the "line in" on a PC sound card, but passed through an amplification stage before they got there (see **Section 3.1**). Once initialized, the transmitter could be moved around the test area and was tracked by the software. The transmitter did not require an external power amplification stage, as it was being driven by the "speaker out" of a sound card.

As seen in **Figure 6.1**, we required foam padding around the test area to absorb sound that would otherwise reflect off of the nearby computers or refrigerator. We also had to elevate the transmitter about 5 inches off the ground so that the horizontal plane of the speaker was the same as the horizonal plane created by the tops of prototype wooden resonators on the receivers. This was to ensure omnidirectionality of the receivers (see **Section 3.3**). The transmitter used the aluminum reflector described in **Section 3.2.1**.

### 6.1.1 Software

This milestone demonstration exclusively used the MATLAB Data Acquisition Toolbox to control the interface with the PC sound card. (For a general overview of this process, see **Section 5.1.1**.) What follows is a walkthrough of a simplified version of the code used to create Milestone 1.

```
ai = analoginput('winsound');
ao = analogoutput('winsound');
```

Figure 6.1: Milestone 1: two Rx tracking one Tx.

These two commands create analog input and analog output objects associated with the I/O subsystems on a PC sound card.

```
addchannel(ai, [1]);
addchannel(ao, [1 2]);
```

One analog input channel and two analog output channels are created, for the 2 transmitters and 1 receiver necessary for the demonstration.

```
set([ai ao],'TriggerType','Manual');
```

This sets the trigger type to "manual," meaning that once the device is armed, a software ***trigger*** command has issued before any data acquisition or waveform generation may begin.

```
set(ai, 'SampleRate', 44100);
set(ai, 'BitsPerSample', 16);
set(ai,'SamplesPerTrigger',inf);
set(ao, 'SampleRate', 44100);
set(ao, 'BitsPerSample', 16);
set(ao, 'RepeatOutput', runloop+4);
```

The first three lines tell the sound card drivers to sample incoming analog audio data at 44.1 kHz with 16 bits of resolution continuously until a ***stop*** command is issued. The last three lines tell the sound card drivers to generate any waveforms at 44.1 kHz, with 16 bits of resolution, and to repeat the waveform being generated *runloop+4* times, where *runloop* is an integer set earlier in the demonstration code.

```
putdata(ao,data_out);
start([ai ao]);
trigger([ai ao]);
```

The ***putdata*** function queues the contents of the *data_out* vector into the data acquisition engine so that it can be directly accessed by the driver software, in this case for the purpose of analog output. The ***start*** function arms the analog input and output subsystems of the hardware, and the ***trigger*** function causes the subsystems to begin data acquisition and waveform generation, respectively.

```
while [...]
data = getdata(ai,2^power);
[process data]
```

```
end
```

The main loop of the program occurs immediately after the device is armed and triggered. For each iteration of the ***while*** loop, ***getdata*** is called, which writes $2^{power}$ samples from the data acquisition engine to the vector *data*. The *data* vector is then manipulated to the specifications of the location algorithm.

```
stop([ai ao]);
delete([ai ao]);
```

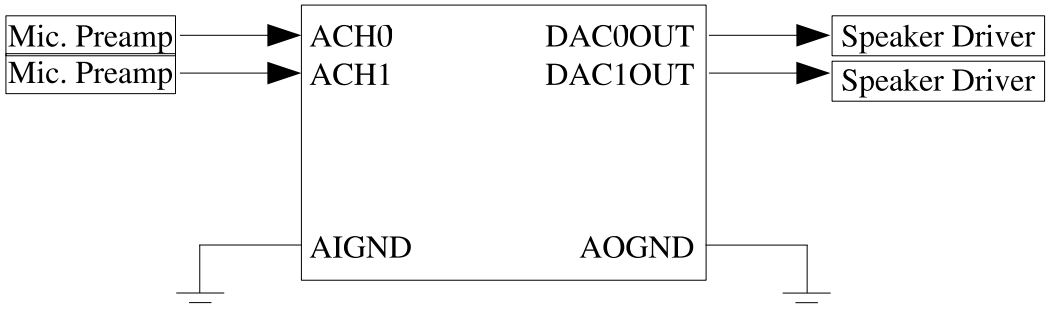
At this point, ***stop*** disarms the subsystems and ***delete*** removes the device objects from the MATLAB workspace.

## 6.2 Milestone 2

The second iteration of the indoor geolocator used one receiver and two transmitters, which is the same configuration as Resnick's original demonstration (**Section 1.3**). The two transmitters remained at fixed locations, each on an opposite diagonal corner of a 22 inch square. The transmitters were oriented facing toward the initialization point where the microphone was located during the initialization loop at the start of code runtime. The microphone was moved about the test area once the initialization loop was completed. This demonstration was effectively the same as Resnick's demonstration, except all signals are processed through the DAQCard 6062-E instead of the sound card.

Also of note is that at the time of this milestone, the hardware was configured with the option to run with two receivers and one transmitter. In **Figure 6.2**, one will notice that there are two microphone preamplifiers and two power amplifiers. The

Figure 6.2: Connections to the SCB-68 connector block. Note that the configuration is flexible for a 2 Tx, 1 Rx or 1 Tx, 2 Rx demonstration.




NI DAQCard was configured to run with 2 of one transducer and 1 of the other transducer, so the MATLAB code runs both the 2 Tx, 1 Rx demonstration and the 2 Rx, 1 Tx demonstration.

### 6.2.1 Hardware

The hardware configuration was as follows: the computer was connected to the DAQCard 6062-E data acquisition card via the PCMCIA bus. There was a shielded cable that connected the PCMCIA card to the SCB-68 I/O connector block. The connector block had its analog inputs ACH0 and ACH1 connected to the microphone preamps. The block's analog outputs DAC0 and DAC1 were connected to the speaker amplifiers, which became necessary to drive the transmitter as the DAQCard does not have any onboard power amplification. See **Figure 6.2** for signal connections. Of note is the power supply for the speaker amplifier circuit, a 36V, 10A HP Harrison 6433B DC Power Supply, which was borrowed from the shop. The microphone preamplifier was powered by a 9V battery.

### 6.2.2 Software

Much of the code for interfacing with the hardware systems remained the same between Milestone 1 and Milestone 2. The portability of the Milestone 1 code was due to the fact that every device object created by the Data Acquisition Toolbox has the same set of base properties. The object-oriented nature of the toolbox simply necessitated that the objects were constructed to communicate with a different driver set, in this case the NI-DAQ 6.9.3 drivers. What follows are those changes that were made.

```
ai = analoginput('nidaq',1);
ao = analogoutput('nidaq',1);
```

These two commands create analog input and analog output objects associated with the I/O subsystems on the DAQCard 6062-E, although the commands would be compatible with any NI device using the NI-DAQ drivers.

```
addchannel(ai, [0 1]);
addchannel(ao, [1]);
```

One analog output channel and two analog input channels are created, for the 2 receivers and 1 transmitter necessary for the demonstration. Also of note is that channels on the NI-DAQ device are identified starting from 0, whereas the PC sound card channels (as seen in **Section 6.1.1**) are indentified starting from 1.

```
set(ai.Channel,'InputRange',[0 10]);
set(ai,'InputType','SingleEnded');
set(ai,'ChannelSkew',2.265e-005/2);
```

The first line of code sets the voltage range expected at the inputs of either channel to between 0 V and 10 V. The analog inputs are then configured to referenced single-ended (RSE), as described in **Section 4.2.1**. Finally, the *channelSkew* parameter is set to an empirically determined value. The value of *channelSkew* determines the time between sampling of consecutive channels during a channel scan. In terms of hardware control, *channelSkew* simply dictates the amount of time between pulses of the CONVERT* signal during a STARTSCAN cycle, as outlined in **Section 4.2.2**.

```
ZZZ = zeros(3,1);

putdata(ao,ZZZ);
```

These last lines occur just before the device objects are disarmed and deleted, and were added as the result of a problem encountered during initial device tests.

We conducted our first tests of the DAQCard not using the OFDM signal as our output signal, but rather we used a 1 kHz sinusoid. We quickly discovered that, upon loading MATLAB, the first time we would attempt to run a program whose output waveform was a sinusoid, everything would work as predicted. However, the next consecutive run of the program caused significant distortion and overdrive of the outputs.

This distortion was identified as a transient on the output, and after further testing we discovered that disarming and deleting an object does not necessarily reset the state of its outputs. So if the analog output device is disarmed at the crest of generating a sinusoid of a 2 V amplitude, the output remains at 2 V indefinitely after the object is deleted. As a result of this finding, a vector of zeroes was written to the analog output device just before disarmament and deletion, which ensured no distorition due to transients would occur in consecutive runs of the program.

Figure 6.3: Photograph of the Milestone 3 configuration.

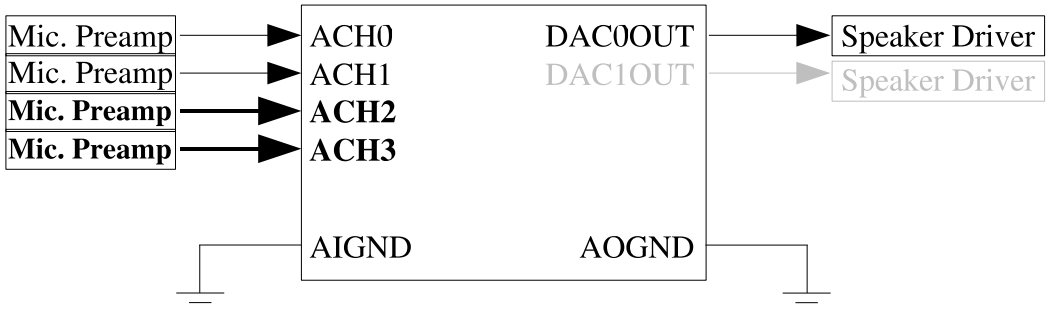
## 6.3 Milestone 3

This iteration of the indoor geolocator was configured for four audio receivers and one audio transmitter. The four receivers remained at fixed locations, the coordinates of which are specified in code. Each microphone was embedded in an acrylic resonator, as described in **Section 3.3**. See **Figure 6.3** for a photograph of the hardware configuration.

### 6.3.1 Hardware

Not very many changes to the hardware were necessary for this milestone; due to the inherent extensibility of our system, most of the changes were made in software. The exception to this was the creation of two additional microphone pream-

Figure 6.4: Milestone 3 signal connections. Note the addition of two analog input channels and the removal of an analog output.




plifier circuits and resonators (necessary for the addition of new channels). See **Figure 6.4** for connections.

### 6.3.2 Software

At this point in the project, the code for the locator algorithm had been extended to accept a variable number of input channels (previously it was hard-coded to accept 2 inputs). A new system-wide variable named *N_out* was added to the software. This variable simply specified the number of input channels to use. Not only was *N_out* passed to the locator algorithm, but it was also used when physically adding channels to the hardware. With this implemented, the locator algorithm would, by definition, expect the same number of input channels as are allotted in hardware.

Also, the *channelSkew* value divided by 2. This reflects the fact that this milestone included 2 times as many receivers as were included in the last milestone, and thus the channels needed to be multiplexed twice as fast.

## 6.4 Other Revisions

Several features were added to the demonstration software in the course of the project that were not directly related to improving position estimation.

### 6.4.1 HTML Statistics Logger

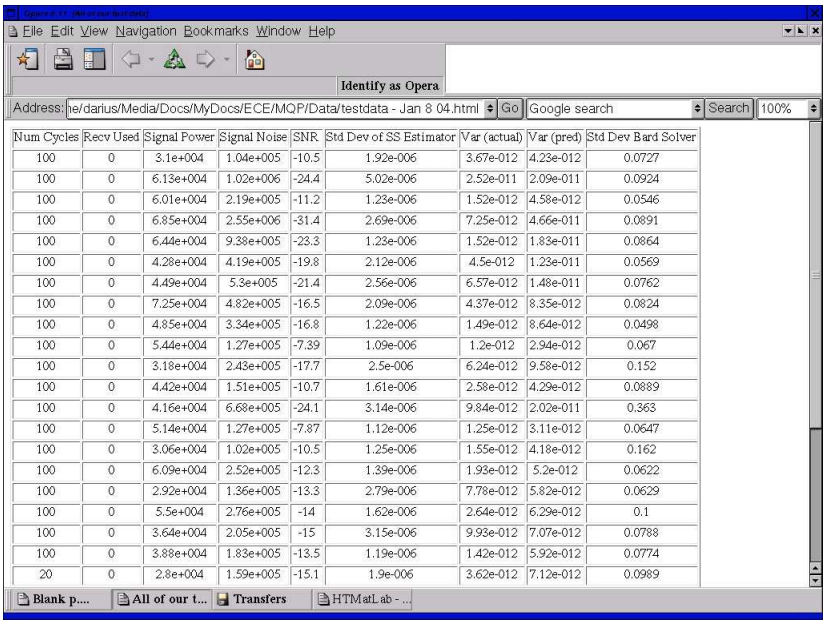
Once the audio demonstration was complete, we began collecting statistical data during each run of the demonstrator. At first, the program simply output the relevant statistical data to the MATLAB command window at the end of each run. Eventually we determined that an automatically logged record of all statistics would be desirable so that we could easily calculate performance averages over large numbers of runs without having to manually record all the data ourselves.

To solve this problem, a piece of code was appended to the end of the main audio demonstration code which put all of the relevant statistical data in a single matrix and used the MATLAB command ***dlmwrite*** to append the matrix to an existing .TXT file which contains the statistical data from any previous runs since the last time statistics were cleared from the file. The .TXT file is then read by ***dlmread***, which puts **all** of the data from all the recorded runs into a single matrix. Finally, an unofficial tool called the MATLAB HTML Toolbox[1] takes the contents of that matrix and outputs it to a file in the form of an HTML table. HTML header data is manually added using MATLAB's file I/O functionality.

As a result of this, every time the duplex4 code is run, a row is appended to a table in an HTML document with the newest statistics. The result is easy to read and interpret (see **Figure 6.5**).

[1] http://www.vision.ime.usp.br/~casado/matlab/htmltoolbox/

Figure 6.5: Output of the HTML statistics generator.


File Edit View Navigation Bookmarks Window Help

Identify as Opera

Address: he/darius/Media/Docs/MyDocs/ECE/MQP/Data/testdata - Jan 8 04.html Go Google search Search 100%

| Num Cycles | Recv Used | Signal Power | Signal Noise | SNR | Std Dev of SS Estimator | Var (actual) | Var (pred) | Std Dev Bard Solver |
|---|---|---|---|---|---|---|---|---|
| 100 | 0 | 3.1e+004 | 1.04e+005 | -10.5 | 1.92e-006 | 3.67e-012 | 4.23e-012 | 0.0727 |
| 100 | 0 | 6.13e+004 | 1.02e+006 | -24.4 | 5.02e-006 | 2.52e-011 | 2.09e-011 | 0.0924 |
| 100 | 0 | 6.01e+004 | 2.19e+005 | -11.2 | 1.23e-006 | 1.52e-012 | 4.58e-012 | 0.0546 |
| 100 | 0 | 6.85e+004 | 2.55e+006 | -31.4 | 2.69e-006 | 7.25e-012 | 4.66e-011 | 0.0891 |
| 100 | 0 | 6.44e+004 | 9.38e+005 | -23.3 | 1.23e-006 | 1.52e-012 | 1.83e-011 | 0.0864 |
| 100 | 0 | 4.28e+004 | 4.19e+005 | -19.8 | 2.12e-006 | 4.5e-012 | 1.23e-011 | 0.0569 |
| 100 | 0 | 4.49e+004 | 5.3e+005 | -21.4 | 2.56e-006 | 6.57e-012 | 1.48e-011 | 0.0762 |
| 100 | 0 | 7.25e+004 | 4.82e+005 | -16.5 | 2.09e-006 | 4.37e-012 | 8.35e-012 | 0.0824 |
| 100 | 0 | 4.85e+004 | 3.34e+005 | -16.8 | 1.22e-006 | 1.49e-012 | 8.64e-012 | 0.0498 |
| 100 | 0 | 5.44e+004 | 1.27e+005 | -7.39 | 1.09e-006 | 1.2e-012 | 2.94e-012 | 0.067 |
| 100 | 0 | 3.18e+004 | 2.43e+005 | -17.7 | 2.5e-006 | 6.24e-012 | 9.58e-012 | 0.152 |
| 100 | 0 | 4.42e+004 | 1.51e+005 | -10.7 | 1.61e-006 | 2.58e-012 | 4.29e-012 | 0.0889 |
| 100 | 0 | 4.16e+004 | 6.68e+005 | -24.1 | 3.14e-006 | 9.84e-012 | 2.02e-011 | 0.363 |
| 100 | 0 | 5.14e+004 | 1.27e+005 | -7.87 | 1.12e-006 | 1.25e-012 | 3.11e-012 | 0.0647 |
| 100 | 0 | 3.06e+004 | 1.02e+005 | -10.5 | 1.25e-006 | 1.55e-012 | 4.18e-012 | 0.162 |
| 100 | 0 | 6.09e+004 | 2.52e+005 | -12.3 | 1.39e-006 | 1.93e-012 | 5.2e-012 | 0.0622 |
| 100 | 0 | 2.92e+004 | 1.36e+005 | -13.3 | 2.79e-006 | 7.78e-012 | 5.82e-012 | 0.0629 |
| 100 | 0 | 5.5e+004 | 2.76e+005 | -14 | 1.62e-006 | 2.64e-012 | 6.29e-012 | 0.1 |
| 100 | 0 | 3.64e+004 | 2.05e+005 | -15 | 3.15e-006 | 9.93e-012 | 7.07e-012 | 0.0788 |
| 100 | 0 | 3.88e+004 | 1.83e+005 | -13.5 | 1.19e-006 | 1.42e-012 | 5.92e-012 | 0.0774 |
| 20 | 0 | 2.8e+004 | 1.59e+005 | -15.1 | 1.9e-006 | 3.62e-012 | 7.12e-012 | 0.0989 |

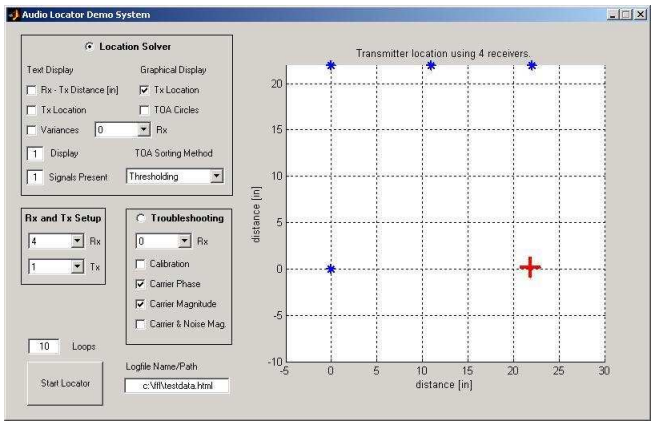
Blank p.... All of our t... Transfers HTMatLab - ...

Additionally, the path and filename of the HTML log file to be written was eventually incorporated into the locator GUI, as seen near the bottom of **Figure 6.6**.

### 6.4.2 Graphical User Interface

A graphical user interface (GUI) was created as an intuitive and user-friendly front-end to the audio locator program. As can be seen in **Figure 6.6**, the GUI not only provides the geometric representation of the fixed receiver locations and the predicted transmitter location, but it also provides the user the ability to start, stop, and configure new test runs. The user can pick from graphical and text display options, choose whether to display statistical data, and troubleshoot the program, among other options.

Figure 6.6: Screen capture of the audio locator GUI.




## 6.5 Results

We performed a series of test runs of the Milestone 3 demonstration, collecting statistical data using the HTML statistics logger. The tests were run with the following parameters:

- 8192-sample length OFDM signal
- 101 carriers
- carrier range of 2.16 kHz to 7.54 kHz

The data in **Table 6.1** are derived from 15 test runs and 1000 cycles of the OFDM signal. At a considerably poor average SNR of about -12 dB, the demonstrator was able to obtain an average positional error standard deviation of about .08 inches, or .2 cm, which is an order of magnitude better than the desired 2 cm of accuracy.

A validation of our method can be found in the TOA accuracy, which is on the order of microseconds. To achieve that level of TOA accuracy using traditional impulse

Table 6.1: Statistical data from 15 runs of the Milestone 3 demonstration at 1000 cycles per run.

| SNR (dB) | Measured TDOA Variance ($sec^2$) | Predicted TDOA Variance ($sec^2$) | Measured Std. Dev. Location (in.) |
|---|---|---|---|
| -12.1 | 1.18e-12 | 5.04e-12 | .0423 |
| -10.1 | 1.08e-12 | 4.02e-12 | .0573 |
| -11.8 | 1.25e-12 | 4.89e-12 | .0527 |
| -14.0 | 1.33e-12 | 6.28e-12 | .0523 |
| -12.1 | 8.81e-13 | 5.08e-12 | .0649 |
| -13.4 | 2.02e-12 | 5.87e-12 | .0597 |
| -11.4 | 8.52e-13 | 4.53e-12 | .0625 |
| -12.4 | 1.29e-12 | 5.25e-12 | .1690 |
| -10.6 | 1.48e-12 | 4.26e-12 | .0624 |
| -10.6 | 2.06e-12 | 4.27e-12 | .0574 |
| -13.1 | 4.50e-12 | 5.69e-12 | .0860 |
| -8.79 | 4.79e-12 | 3.45e-12 | .0599 |
| -10.4 | 1.24e-12 | 4.18e-12 | .0709 |
| -11.1 | 1.57e-12 | 4.51e-12 | .0589 |
| -11.5 | 1.89e-12 | 4.73e-12 | .0605 |

ultrawideband ranging, a microsecond-width pulse would need to be generated, with signal power concentrated at about 1 MHz plus any harmonics. That we managed similar TOA accuracy using signals at frequencies less than 10 kHz is a testament to the utility of our solution method.

# Chapter 7

# RF Demonstration

After completing the third audio demonstration milestone and proving to our satisfaction that the algorithm works at audio frequencies, we decided that it was time to test our system in the RF range.

Switching to RF meant not only a change in the frequency of signal transmission, but also a change in bandwidth for the signal. As mentioned in **Section 2.5**, the wavelength of our audio signals in air correspond physically to the wavelength of certain RF signals in vaccuum. As illustrated in **Table 2.1**, the approximately 5 kHz of bandwidth we use in the audio range corresponds to approximately 5 GHz of bandwidth in the RF range.

While jumping to a 5 GHz bandwidth signal may have been desirable, it certainly was not feasible, as our DAQCard has a maximum analog I/O transfer rate of 500 kHz. We decided to test the limits of the card by attempting to construct a demonstration using a 250 kHz-wide OFDM signal.

We encountered a number of problems in our implementation of the high-bandwidth system. Two problems discussed herein include the physical transmission of such

signals and the signal processing capability of the data aquisition hardware and software.

## 7.1 Terk Leapfrog LF-30S

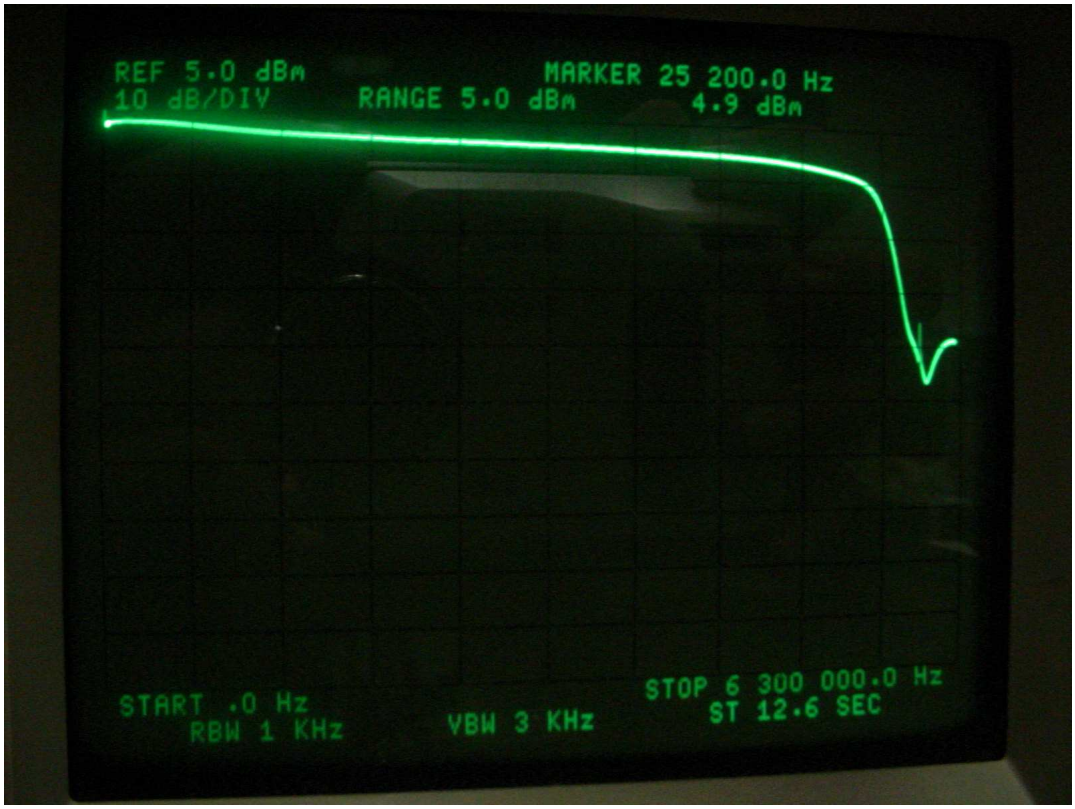
In order to transmit and receive a 500 kHz bandwidth signal, we needed transducers with appropriate characteristics. Although an RF transmitter for our application was being developed at the same time by another group associated with the DOJ-sponsored efforts, we needed something immediately so that we could test the robustness of our signal when its frequency is stepped up, transmitted at RF, stepped down, and then analyzed by MATLAB. We looked at several commercial RF transmitters until we found one that we felt would meet our specifications.

The Terk Leapfrog LF-30S wireless audio/video transmitter and receiver system is a commercially available device whose purpose is to transmit A/V signals wirelessly around a home at 2.4 GHz. The LF-30S broadcasts using FM modulation at frequencies from 2.4 GHz to 2.48 GHz. The system can be set to use one of 4 selectable channels in its broadcast range.

During tests, we discovered that the 2.4 GHz carrier frequency of the wireless IEEE 802.11b network in our building interfered with the operations of the LF-30S, but by changing the transmission channel we were able to discover a channel where interference was negligible.

Before using the LF-30S, we needed to find its transmission bandwidth. To measure the bandwidth, we connected the tracking oscillator output from a spectrum analyzer to the input of the video link transmitter, then connected the video link receiver output to the input of the spectrum analyzer. We obtained the frequency

Figure 7.1: Frequency response of the LeapFrog A/V transceiver system.




response shown in **Figure 7.1**. As can be seen in the figure, the LF-30S has a 6 MHz tranmission bandwidth, far exceeding our 250 kHz bandwidth requirement.

Unfortunately, once the LF-30S was integrated into the system, we discovered that the DAQCard was incapable of sampling the waveform at the rates we desired.

## 7.2 Data Acquisition

Although the DAQCard 6062-E is specified as being capable of providing 500 ksamples/sec of continuous I/O, actually obtaining that level of throughput for our RF signal proved to be far more difficult than initially imagined.

### 7.2.1 Throughput of NI-DAQ 6.9.3 Drivers

The first roadblock in the process of developing a 500 ksample/sec acquisition system was the implementation of the NI-DAQ 6.9.3 driver function set (see **Section 5.2** for a detailed description of the software driver functionality). All of our interfacing with the DAQCard is done through this particular set of driver functions, regardless of whether or not we use the MATLAB front-end or we access the C functions in the DLLs that are called from MATLAB using the **loadlibrary** function. In order to initiate continuous I/O with an external clock source, we needed to use the **DAQ_DB_Transfer** function.

Unfortunately, it was empirically determined by changing the frequency of the external sampling clock that the function would fail at rates above roughly 200kHz of thoughput. If the sampling rate is set (through external hardware) to be any higher than that, the computer stalls on the **DAQ_DB_Transfer** call. A conjecture which was confirmed by National Instruments engineers is that this was due to the use of a **while** loop in the **DAQ_DB_Transfer** call which slowed down the entire continuous I/O process.

We could not alter the drivers themselves to remove the **while** loop, as they came as precompiled libraries.

### 7.2.2 NiDAQmx Testing

National Instruments engineers informed us that a new version of NI-DAQ, called NiDAQmx, offered roughly a 20-fold performance increase over our current driver set. Since all we required was a 3-fold increase, even a conservative performance increase in the new driver set would have been satisfactory.

Unfortunately, any performance gains in the new driver set would never be seen. After two weeks of testing the new driver set, we discovered that the **loadlibrary** function embedded in MATLAB 6.5 R13SP1 could not call the C++ functions provided by the new NiDAQmx. This is because the new drivers use 64-bit integers (**int64, uint64, int64Ptr, and uint64Ptr**), with which MATLAB is not yet fully compatible.

## 7.3 Counter-Trigger Synchronization Method

The fact that the NiDAQmx drivers were not compatible with MATLAB made it seem as though we had hit a roadblock that we could not navigate past. Several weeks were spent wrestling with this problem, until we took a step back and looked at the way the entire project functioned. During a brainstorming session, it occurred to us that we are only updating position information on the user end from 2 to possibly 4 times a second, hence we do not need to continuously acquire data at a rate higher than a single 8192-sized sample block 2 to 4 times a second. If we could figure out a way to aquire discontinuous 8192-sample blocks such that each sampled block occurred at the exact same time offset with respect to the continuous and repeating transmitted OFDM signal, then we could replace continuous double-buffered sampling with highly synchronous single-buffered one-shot sampling. With single-buffered sampling, we could use the original NI-DAQ 6.9.3

drivers and acquire a signal at a sampling frequency of up to 500 kHz.

It was at this point that we came up with our counter-trigger solution, which completely bypassed the need for a continuous software loop by handling almost all of the processing in hardware, onboard the DAQCard. A test system was built which proved that the single-buffering solution is feasible.

### 7.3.1 Counter-Trigger Test Architecture

As illustrated in **Figure 7.2**, the architecture of the counter-trigger system consists of two computers. The DAQ board attached to computer A continuously outputs its preloaded buffer, which consists of one period of the OFDM signal and is 8192 samples in length. In addition to generating the waveform, it also generates the UPDATE* signal. When configured as "an output, UPDATE* reflects the actual update pulse that is connected to the DACs"[1]. The OFDM signal is generated on the DAC0 line, and is connected to the analog input channel 1 line ACH1 on computer B's DAQCard. The UPDATE* signal is asserted to the PFI5/UPDATE* line, which is connected to the PFI2/CONVERT* line on computer B's DAQCard.

Although the ADC on computer B is continually reading data at its input, and is clocked at the same rate as computer A is generating data, it does not begin analog-to-digital conversion until it receives a start trigger. The start trigger comes from the general-purpose counter (GPCTR0) which is configured to generate a pulse train.

[1] *DAQCard 6062-E User Manual*. National Instruments, June 2002.

### 7.3.2 Pulse Trains

The general-purpose counters on the DAQCard 6062-E have a set of control registers which can be written to cause different behaviors, and any of these counters can be configured to generate a pulse train. A pulse train needs to know three things to function: a valid ***source*** identifier, an integer ***hiCount***, and an integer ***loCount***.

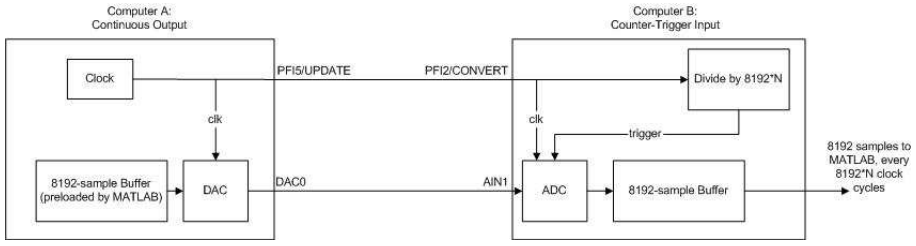
The counter is triggered to count on the rising edge of the signal specified by ***source***. The counter's output is high (+5 V) until it reaches the value specified by ***hiCount***, at which point the counter resets and the output is set to low (0 V). The output remains low until the counter reaches the ***loCount*** value, when the counter resets and the output is inverted once again.

What this means is that we can set a general purpose counter so that ***source*** is the sampling clock of the OFDM waveform and so that ***hiCount*** = ***loCount*** = 8192*N. If the counter ouput is then tied to the ADC trigger and the trigger is set as a low-to-high signal, which means that single-buffer sampling would occur once every 8192*2N samples are output/read (the output/read count is the same because the clocks on both the continuous output computer A the counter-trigger input computer B and synchronous). N can be any natural number, and can be tweaked to achieve the desired number of single-buffer reads per second.

### 7.3.3 Verification of Counter-Trigger Method

To verify that our new sampling method worked, data was collected from a system set up in the manner depicted in **Figure 7.2**. If the time offset of the data sampled on computer B was truly constant relative to the period of the repeating waveform output by computer A, then if we take multiple 8192-sample blocks of data and run them as separate, simultaneous signals through the state-space estimator to obtain

Figure 7.2: Block diagram of counter-trigger control configuration.




a list of TDOA values, those TDOAs should be very close to 0.

Performing this verification, the TODAs between the noncontinuously sampled single-buffer blocks averaged about .1 ns, which was an order of magnitude better than we had hoped for. We had successfully sampled the OFDM signal synchronously at a sampling rate of 500 kHz.

# Chapter 8

# Conclusions

Over the course of a year, this project resulted in the successful migration of the locator demonstration from a PC sound card to the NI DAQCard 6062-E. In the course of this migration, we were not only able to extend the functionality of the demonstration itself, but we were also able to gather performance data and run tests to determine the feasibility of further extension of the system on the DAQCard platform.

## 8.1 Future Work

Work which can be completed in the near future includes a working RF demonstration. Since the DAQCard has been shown to be capable of noncontinuous synchronous single-buffer analog I/O at 500 kHz, it should be possible to implement a locator demo operating at 500 kHz using 250 kHz of bandwidth via the Leapfrog LF-30S tranceiver set.

Possibly before an RF demonstration is completed, and certainly before moving

up to higher carrier frequencies and bandwidths, it will be desirable to have each transmitter and reference node operating on its own laptop computer. This requires synchronization of all reference nodes to each other in order to assure accurate TDOA extraction, which can be achieved by synchronizing the GPCTR control signals for our counter-trigger architecture.

Finally, a three-dimensional demonstration will be needed. Although some preliminary work has been done with the audio demonstrator to test feasibility, the directionality of the speaker/microphone coupling system made this rather difficult to conduct. With the completion of the RF system it should be possible to explore full 3D operation with greater ease.

Once the demonstration is operating at high frequencies with a high bandwidth, complete reference node synchronization, and in three dimensions, we will have a model that provides a powerful analog to the envisioned Precision Personnel Locator.

# Appendix